CE343: Software Engineering

Unit-1

Unit- 1

1. Introduction to Software and Software Engineering
   - The Evolving Role of Software, Software: A Crisis on the Horizon and Software Myths
   - Software Engineering: A Layered Technology
   - Software Process Models, The Linear Sequential Model, The Prototyping Model, The RAD Model, Evolutionary Process Models

What is Software?

Software refers to a set of instructions, programs, or data that enables a computer or a computing device to perform specific tasks.

There are two primary categories of software:

1. System Software:
   - Examples of system software include Microsoft Windows, macOS, Linux, and device drivers for printers or graphics cards.
2. Application Software
   - Examples include Microsoft Word, Google Chrome, Photoshop, and video games.

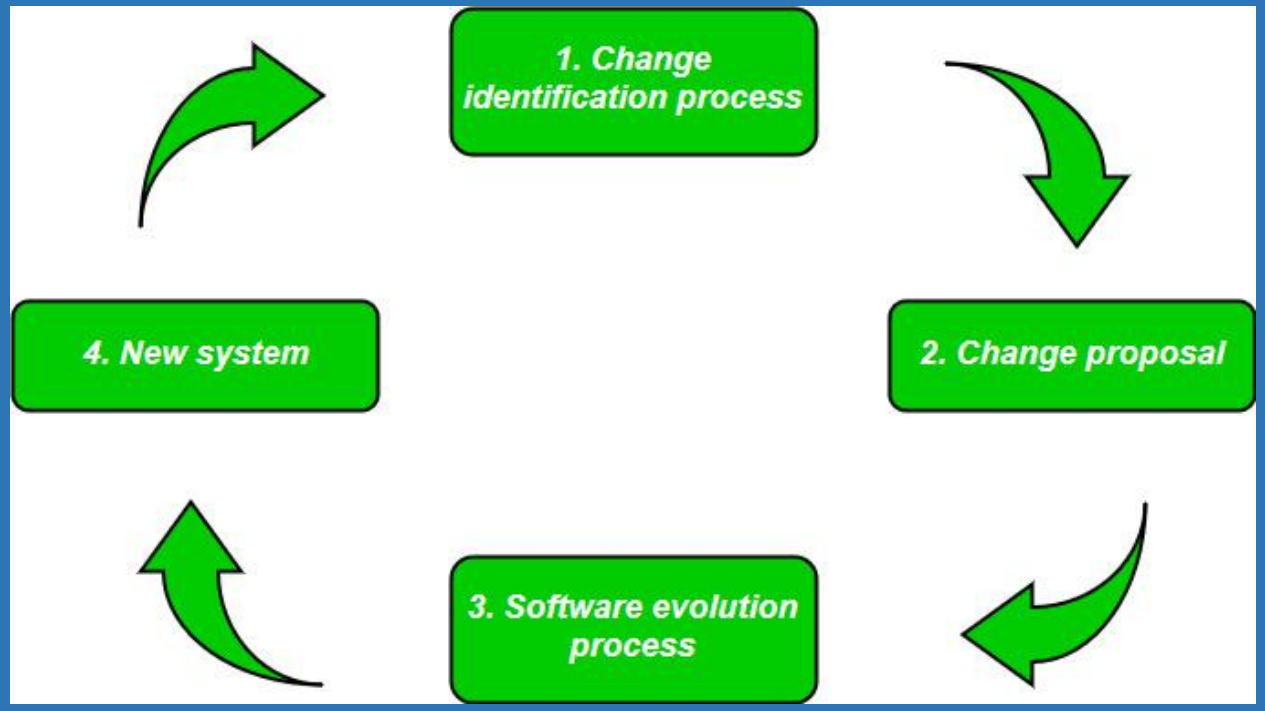
What is Software?

Software can be further classified based on its distribution and use:

- **Proprietary Software:** Proprietary or commercial software is developed by a company or an individual and is protected by copyright.(e.g. MS Office, Adobe Photoshop)
- **Open Source Software:** Open source software is released with a license that allows users to view, modify, and distribute the source code.
- **Freeware and Shareware:** Freeware is software that is available for use at no cost, while shareware is typically free to try but requires payment for continued use or access to additional features.
- **Web Applications**
- **Embedded software**
- **Artificial intelligence software**

Evolving Role of Software

Software Evolution is a term which refers to the process of developing software initially, then timely updating it for various reasons, i.e., to add new features or to remove obsolete functionalities etc. The evolution process includes fundamental activities of change analysis, release planning, system implementation and releasing a system to customers.



Evolving Role of Software

The key aspects that highlight the evolving role of software in software engineering:

- Increased Complexity of Software Systems
- Shift to Agile and DevOps
- Focus on User-Centric Design
- Rise of Cloud Computing
- Machine Learning and Artificial Intelligence Integration
- Security-First Approach
- Open Source and Collaboration
- Focus on Microservices Architecture
- Focus on Quality Assurance and Testing

Evolving Role of Software

Case Study: Revolutionizing Healthcare with Evolving Software Solutions

Objectives:

- Enhance Patient Experience
- Improve Operational Efficiency
- Ensure Data Security and Compliance
- Facilitate Collaborative Healthcare

Phase 1: Legacy System Migration (Year 1-2): Migrate to Cloud environment

Phase 2: Patient-Centric Applications (Year 2-3): Develop portal and mobile applications

Phase 3: AI-Driven Healthcare (Year 3-4): Integrate AI algorithms for predictive analytics etc..

Phase 4: Continuous Improvement and Innovation (Year 4+): embracing emerging technologies such as blockchain for secure data sharing and interoperability.

Software Crisis

- The difficulty of writing useful and efficient computer programs in the required time.
- The software crisis was due to using the same workforce, same methods, and same tools even though rapidly increasing software demand, the complexity of software, and software challenges.

Factors Contributing to Software Crisis:

- Poor project management.
- Lack of adequate training in software engineering.
- Less skilled project members.
- Low productivity improvements.

Software Myths

Three Types Of Software Myths

- Management Myths
- Customer Myths
- Practitioners Myth

Software Myths: Management

Myth1: We have all the standards and procedures available for software development.

Fact:

- Software experts do not know all the requirements for the software development.
- And all existing processes are incomplete as new software development is based on new and different problem.

Myth 2: The addition of the latest hardware programs will improve the software development.

Fact:

- The role of the latest hardware is not very high on standard software development; instead (CASE) Engineering tools help the computer, they are more important than hardware to produce quality and productivity. Hence, the hardware resources are misused.

Software Myths: Management

Myth1: With the addition of more people and program planners to Software development can help meet project deadlines (If lagging behind).

Fact:

- Spend time educating the newcomers
- The newcomers are also far less productive than the existing software engineers.

Software Myths: Customer

Customer myth **occurs due to false expectation** by customers and this can lead customers to become dissatisfied with software developers.

Myth1: Software requirements continually change, but change can be easily accommodated because software is flexible.

Facts:

- When requirements changes are requested early (before design or code has been started), the cost impact is relatively small.
- The cost impact grows rapidly—resources have been committed, a design framework has been established

Software Myths: Practitioner's

Developers often have to work under pressure from their managers to finish software projects quickly and with limited resources.

Myths 1: **They believe that their work has been completed with the writing of the plan.**

Fact:

It is true that every 60-80% effort goes into the maintenance phase (as of the latter software release). Efforts are required, where the product is available first delivered to customers.

Myth 2: **There is no other way to achieve system quality, until it is "running".**

Facts:

Systematic review of project technology is the quality of effective software verification method. These updates are quality filters and more accessible than test.

Software Myths: Practitioner's

Myth 3: **An operating system is the only product that can be successfully exported project.**

Fact:

A working system is not enough, the right document brochures and booklets are also required to provide guidance & software support.

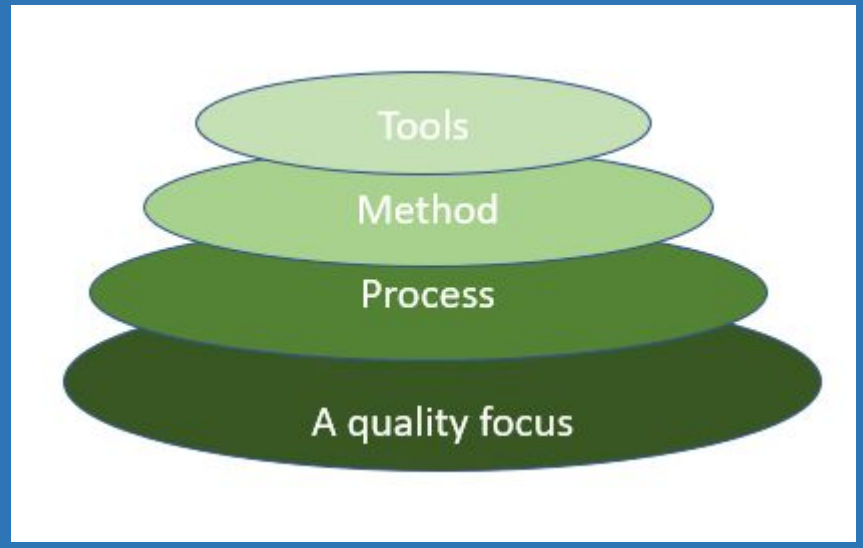
Myth 4: **Engineering software will enable us to build powerful and unnecessary document & always delay us.**

Fact:

Software engineering is not about creating documents. It is about creating a quality product. Better quality leads to reduced rework. And reduced rework results in faster delivery times

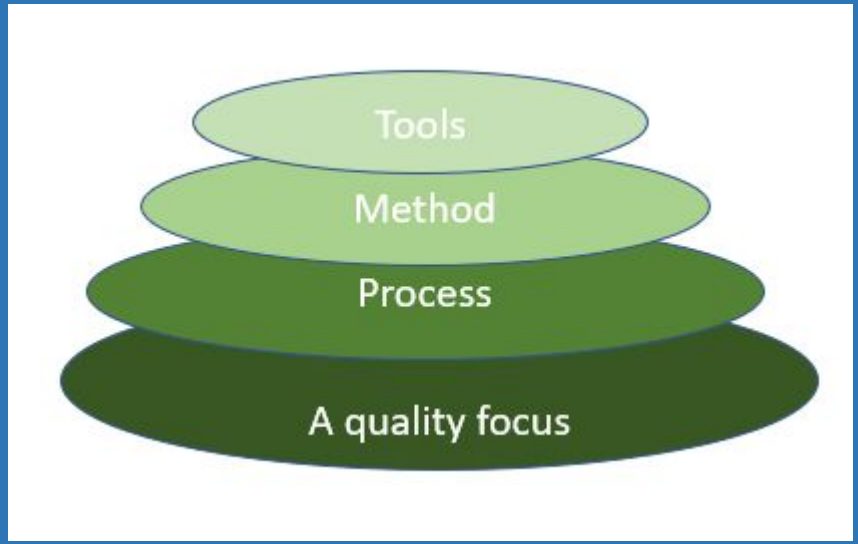
Software Engineering: Layered Technology

1. **A quality focus:** It defines the continuous process improvement principles of software. It provides integrity that means providing security to the software so that data can be accessed by only an authorized person.
2. **Process:** It is the foundation or base layer of software engineering. It is key that binds all the layers together which enables the development of software before the deadline or on time.



Software Engineering: Layered Technology

3. Method: During the process of software development the answers to all “how-to-do” questions are given by method. It has the information of all the tasks which includes communication, requirement analysis, design modeling, program construction, testing, and support.



4. Tools: Software engineering tools provide a self-operating system for processes and methods. Tools are integrated which means information created by one tool can be used by another.

Software Process Models

What is Software Process?

“ The methods and techniques used to develop and maintain software.”

The goal of a software process model is to provide guidance for **controlling and coordinating the tasks to achieve the end product objectives as effectively as possible**.

What is Software Process?

A model will define the following:

- The tasks to be performed
- The input and output of each task
- The pre and post-conditions for each task
- The flow and sequence of each task

Factors in choosing a software process

Project requirements:

- Will the user need to specify requirements in detail after each iterative session?
- Will the requirements change during the development process?

Project size: Larger projects mean bigger teams

Project complexity : Complex projects may not have clear requirements. The requirements may change often, and the cost of delay is high

Cost of delay: Is the project highly time-bound with a huge cost of delay, or are the timelines flexible?

Factors in choosing a software process

Customer involvement: Do you need to consult the customers during the process? Does the user need to participate in all phases?

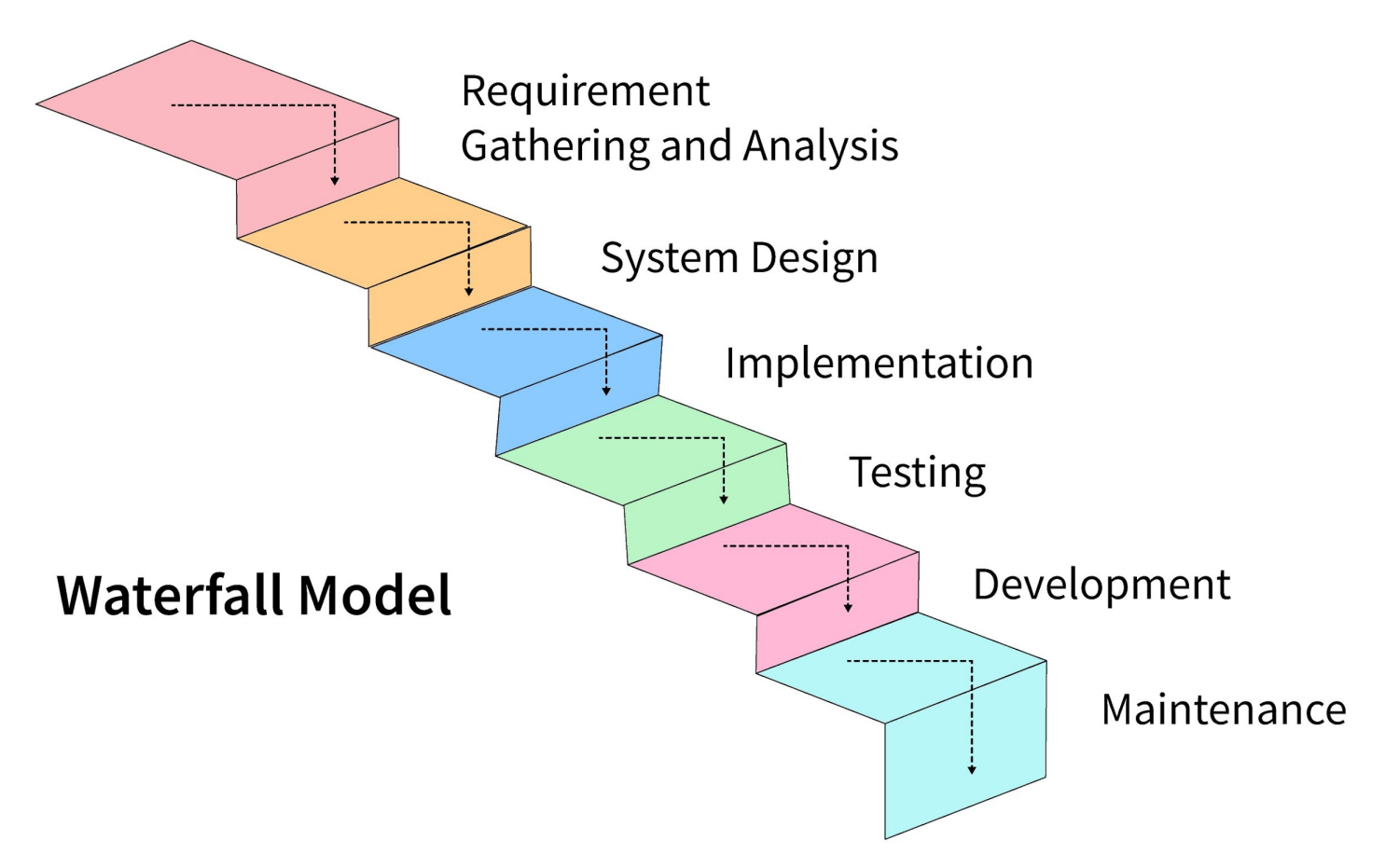
Familiarity with technology: This involves the developers' knowledge and experience with the project domain, software tools, language, and methods needed for development.

Project resources: This involves the amount and availability of funds, staff, and other resources.

Waterfall Model (Linear Sequential)

The waterfall model was first introduced by **Dr. Winston W. Royce** in a paper titled "**Managing the Development of Large Software Systems**," which was presented at a conference in 1970. Dr. Royce outlined the model as a sequential and non-iterative approach to software development.

Waterfall Model (Linear Sequential)



Sequential Phases of the Classical Waterfall Model

Waterfall Model (Linear Sequential)

1. **Feasibility Study :** The main goal of this phase is to determine whether it would be financially and technically feasible to develop the software.
2. **Requirements Analysis and Specification:** The aim of the requirement analysis and specification phase is to understand the exact requirements of the customer and document them properly. This phase consists of two different activities.
    1. **Requirement gathering and analysis:**
        - Firstly all the requirements regarding the software are gathered from the customer and then analyzed it.
        - The goal of the analysis part is to remove incompleteness and inconsistencies.

Waterfall Model (Linear Sequential)

Example of incomplete requirement:

Original Requirement: "The student grading system should be user-friendly.“

Revised Requirement: "The student grading system should allow instructors to input grades easily through a web-based interface. It should include features such as dropdown menus for grade selection, error-checking to prevent input mistakes, and a user guide accessible from the main dashboard.“

Original Requirement: "The system should have fast response times.“

Revised Requirement: "The system should respond to user interactions within 2 seconds, measured from the time a user clicks a button or initiates an action to the time the corresponding response is displayed on the screen."

Waterfall Model (Linear Sequential)

Example of Inconsistent Requirements:

Original Requirements:

Requirement 1: "The system should support both online and offline modes of operation."

Requirement 2: "The system must require an active internet connection for all functionality.“

Revised Requirements:

Requirement 1: "The system should support both online and offline modes of operation, allowing users to access and perform basic functions even when not connected to the internet."

Requirement 2: "Certain advanced features, such as real-time data synchronization and remote collaboration, may require an active internet connection."

Waterfall Model (Linear Sequential)

Example of Inconsistent Requirements:

Original Requirements:

Requirement 1: "The system should prioritize data security and encryption for user information."

Requirement 2: "User data should be easily accessible for customer support representatives to provide efficient assistance.“

Revised Requirements:

Requirement 1: "The system should prioritize data security and encryption for user information, ensuring that sensitive data is protected from unauthorized access."

Requirement 2: "Customer support representatives should have controlled access to user data, with the ability to retrieve information needed for assistance while adhering to strict privacy and security measures."

Waterfall Model (Linear Sequential)

2. **Requirement specification:** These analyzed requirements are documented in a software requirement specification (SRS) document. SRS document serves as a contract between the development team and customers. Any future dispute between the customers and the developers can be settled by examining the SRS document.

4. **Design:** The goal of this phase is to convert the requirements acquired in the SRS into a format that can be coded in a programming language. It includes high-level and detailed design as well as the overall software architecture.

5. **Coding and Unit Testing:** In the coding phase software design is translated into source code using any suitable programming language. Thus each designed module is coded. The aim of the unit testing phase is to check whether each module is working properly or not.

Waterfall Model (Linear Sequential)

5. **Integration and System testing:** Integration of different modules is undertaken soon after they have been coded and unit tested.

   System testing consists of three different kinds of testing activities as described below.

   - **Alpha testing:** Alpha testing is the system testing performed by the development team.
   - **Beta testing:** Beta testing is the system testing performed by a friendly set of customers.
   - **Acceptance testing:** After the software has been delivered, the customer performed acceptance testing to determine whether to accept the delivered software or reject it.

Waterfall Model (Linear Sequential)

6. **Maintenance:** Maintenance is the most important phase of a software life cycle. The effort spent on maintenance is 60% of the total effort spent to develop a full software.

   There are basically three types of maintenance.

   - **Corrective maintenance:** It is performed to repair faults that were not found during the product development process.
   - **Perfective Maintenance:** This sort of maintenance is performed to improve the system's functionality depending on the customer's request.
   - **Adaptive Maintenance**: When porting software to a new environment, adaptive maintenance is frequently necessary.

Waterfall Model (Linear Sequential)

Advantages of the Classical Waterfall Model

Easy to Understand: Classical Waterfall Model is very simple and easy to understand.

Individual Processing: Phases in the Classical Waterfall model are processed one at a time.

Properly Defined: In the classical waterfall model, each stage in the model is clearly defined.

Clear Milestones: Classical Waterfall model has very clear and well-understood milestones.

Properly Documented: Processes, actions, and results are very well documented.

Reinforces Good Habits: Classical Waterfall Model reinforces good habits like define-before-design and design-before-code.

Working: Classical Waterfall Model works well for smaller projects and projects where requirements are well understood.

Waterfall Model (Linear Sequential)

Disadvantages of the Classical Waterfall Model

No Feedback Path: In the classical waterfall model evolution of software from one phase to another phase is like a waterfall. It assumes that no error is ever committed by developers during any phase. Therefore, it does not incorporate any mechanism for error correction.

Difficult to accommodate Change Requests: This model assumes that all the customer requirements can be completely and correctly defined at the beginning of the project, but actually customer's requirements keep on changing with time.

No Overlapping of Phases: This model recommends that a new phase can start only after the completion of the previous phase. But in real projects, this can't be maintained.

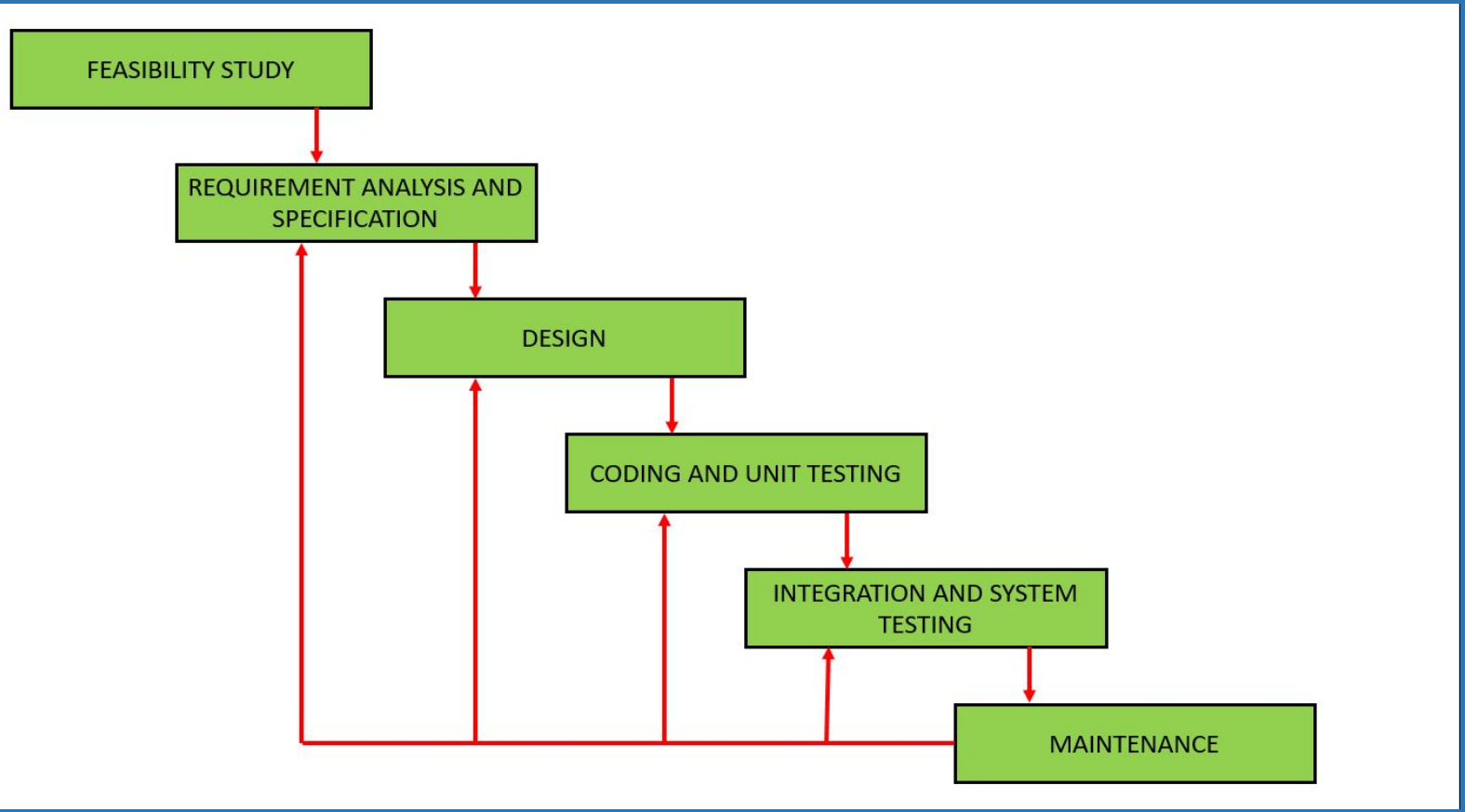
Not Suitable for Complex Projects: Difficult to manage multiple dependencies and interrelated components.

Waterfall Model (Linear Sequential)

When to Use the Classical Waterfall Model

- The clients have a crystal clear understanding of what they want.
- The requirements are significantly less likely to change during the execution of the project.
- The software product being developed is not complicated.
- The tools and technologies to be used for developing the software will not change dynamically.
- The resources required are predictable, and the resources are available to use.

Iterative Waterfall Model



The iterative waterfall model **provides feedback paths from every phase to its preceding phases,** which is the main difference from the classical waterfall model.

Iterative Waterfall Model

When to use Iterative Waterfall Model:

- The prerequisite of being well-defined and comprehended.
- The development team is gaining knowledge about new technologies.
- Certain characteristics and objectives carry a significant chance of failure in the future.
- When there is a need to incorporate customer feedback at every stage - The major requirements are laid down initially; however, as the development process progresses, some functionalities are altered, and additions are suggested.

Iterative Waterfall Model

Advantages of Iterative Waterfall Model :

Feedback Path: It allows correcting the errors that are committed and these changes are reflected in the later phases. (e.g. allowing developers and testers to identify design or functionality faults as quickly as possible, allowing them to take corrective actions.)

Cost-Effective: It is highly cost-effective to change the plan or requirements in the model. Moreover, it is best suited for agile organizations.

Well-organized: In this model, less time is consumed on documenting and the team can spend more time on development and designing.

Iterative Waterfall Model

Advantages of Iterative Waterfall Model :

Quality Assurance: The iterative approach promotes quality assurance by providing opportunities for testing and feedback throughout the development process.

Improved Customer Satisfaction: Customer feedback can be incorporated in every iteration and implemented quickly.

Iterative Waterfall Model

Disdvantages of Iterative Waterfall Model :

Difficult to incorporate change requests Problems relating to the system architecture can arise because all the requirements are not gathered upfront. Design can be changed repeatedly because of defective requirements gathered in the first phase.

Incremental delivery not supported: The full software is completely developed and tested before delivery to the customer. There is no scope for any intermediate delivery.

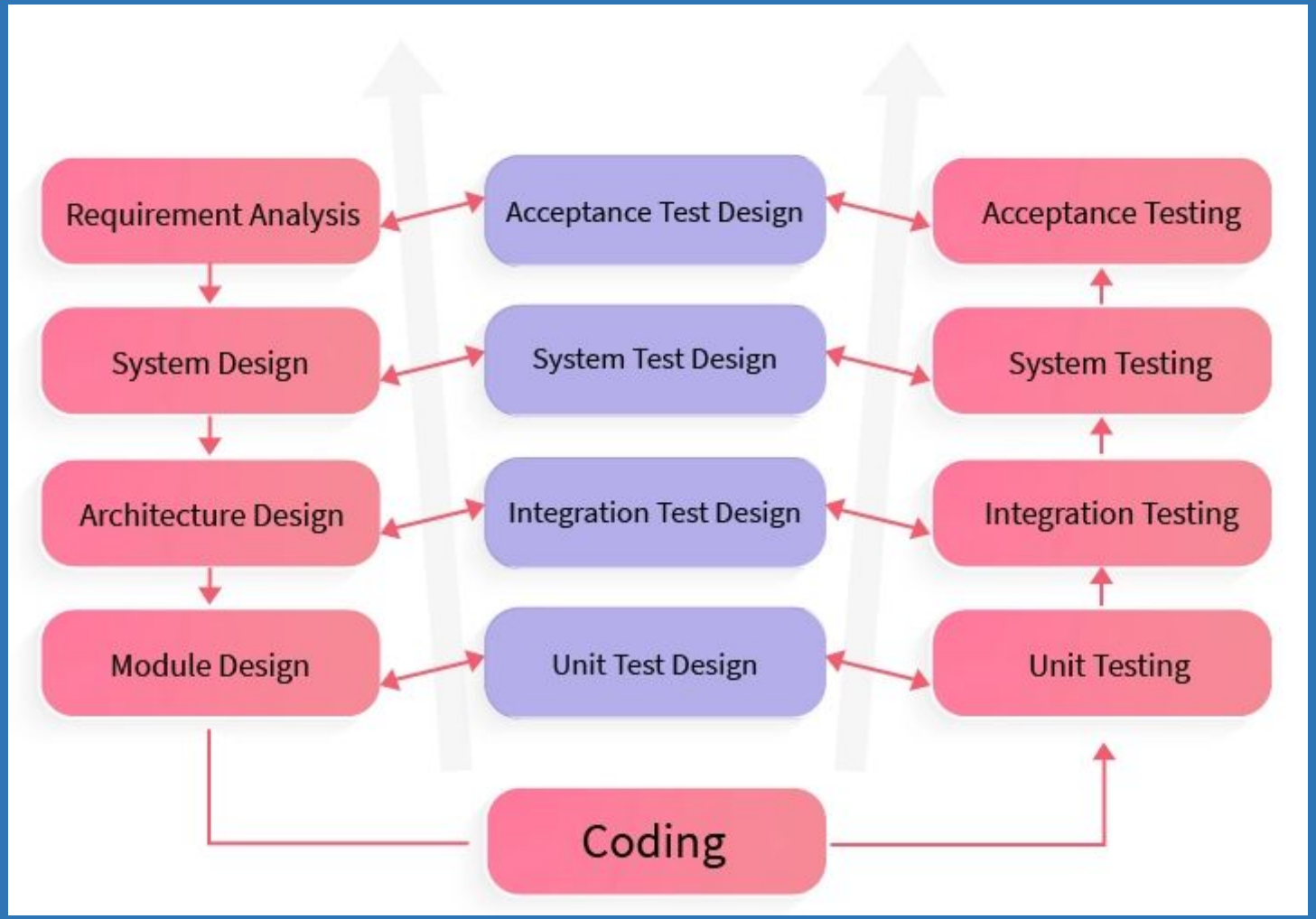
Overlapping of phases not supported Iterative waterfall model assumes that one phase can start after completion of the previous phase, But in real projects, phases may overlap to reduce the effort and time needed to complete the project.

Iterative Waterfall Model

Disadvantages of Iterative Waterfall Model :

Limited customer interactions: Customer interaction occurs at the start of the project at the time of requirement gathering and at project completion at the time of software delivery. These fewer interactions with the customers may lead to many problems as the finally developed software may differ from the customers' actual requirements.

V-Model (Verification and Validation)



V-Model (Verification and Validation)

Verification Phase of V-Model

1. **Business Requirement Analysis:** In this phase, the requirements and needs of customers are understood.
   - What the customer expects from the final software
   - what functionalities customers want,

   This is indeed a very important phase, as many times there is confusion in the mind of both customer and developer regarding the final outcome of the software.

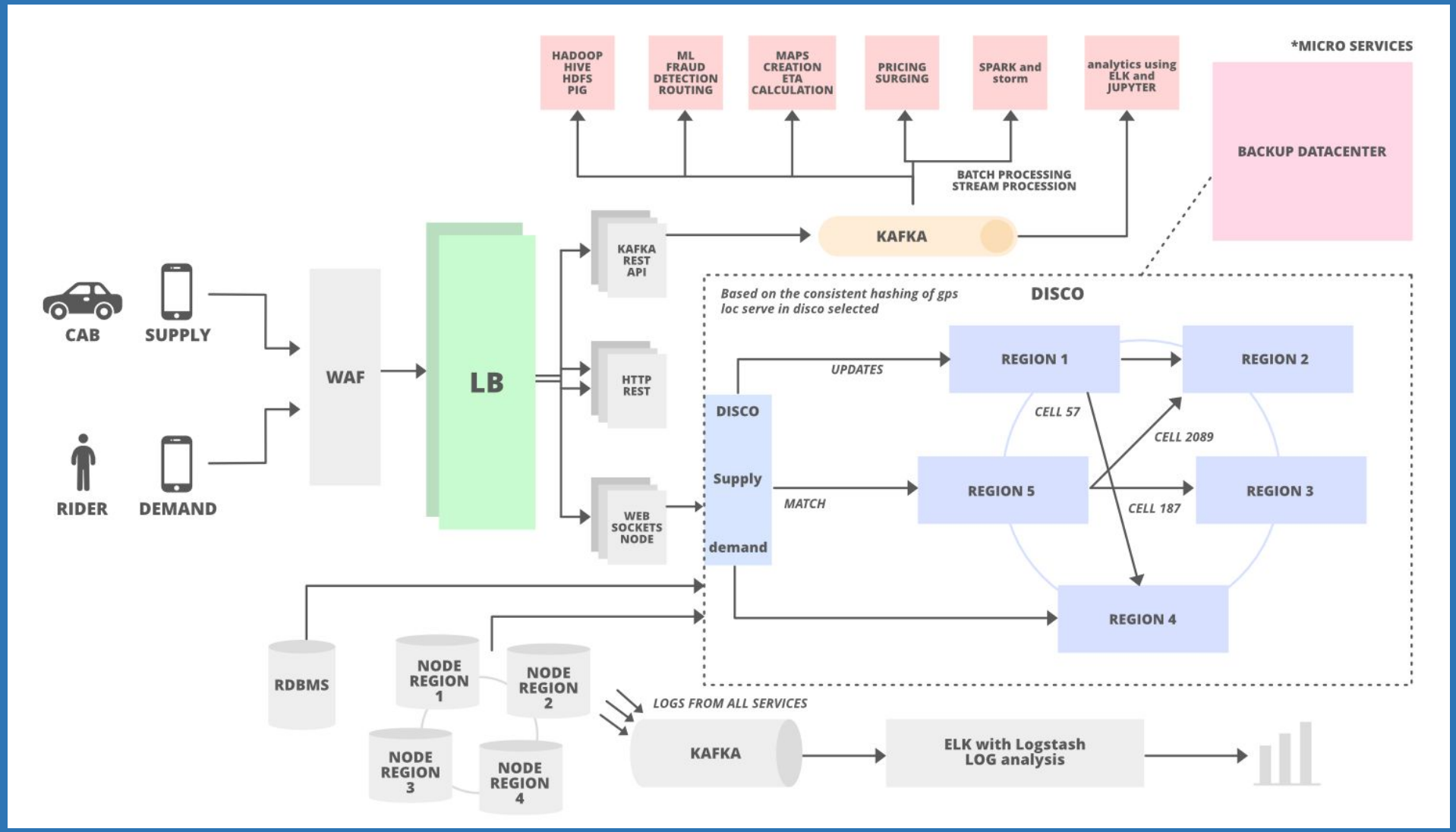
   Acceptance testing is carried out in this phase.

V-Model (Verification and Validation)

Verification Phase of V-Model

2. **System Design:** In this phase, the actual design of the system is decided. After the requirement analysis phase, based on the finalized requirements, the complete design of the system is discussed. It includes the hardware and the communicating setup requirements.
3. **Architecture Design:** This phase is also referred to as High-Level Design(HLD). It consists of various modules, database tables, UML diagrams, etc. In this stage, all the communications between the internal modules of the system and the outer system are understood.
4. **Module Design:** This phase is also known as Low-Level Design (LLD). The compatibility of each internal module and its feasibility is checked. Unit testing is performed in this phase.

V-Model (Verification and Validation)



V-Model (Verification and Validation)

Verification Phase of V-Model

5. **Coding phase:** In the coding phase, coding of the design and specification done in the previous phases is done. This phase takes the most time.

V-Model (Verification and Validation)

Validation Phase of V-Model

1. **Unit Testing:** Unit testing is performed in the module design phase. Here each module goes through the testing by executing the code written for that module. It tests whether each module is able to perform its required functions or not. If not, the bugs are removed so as to produce effective modules.

Example => Test User Authentication:

1. Create unit tests to verify that the login system correctly authenticates users with valid credentials.
2. Include test cases for both successful logins and failed logins (e.g., incorrect username, incorrect password).
3. Ensure that the system responds appropriately to different authentication scenarios.

V-Model (Verification and Validation)

Validation Phase of V-Model

2. **Integration testing** In integration testing, the integration tests created in the architectural design phase are executed. Integration testing ensures that all modules are working well together..

Example:

Integration Test 2: Shopping Cart and Inventory Management

Components Involved:

- Shopping Cart Module
- Inventory Management Module

V-Model (Verification and Validation)

Validation Phase of V-Model

3. **System testing**: The system tests created in the system design phase are executed. System tests check the complete functionality of the system. In this, more attention is given to performance testing and regression testing.

Example:

Performance Testing => Test Scenario: Handling Concurrent Users

Compatibility Testing=> Test Scenario: Cross-Browser Compatibility

Stress Testing=> Test Scenario: Handling Extreme Loads

V-Model (Verification and Validation)

Validation Phase of V-Model

4. **Acceptance testing:** In acceptance testing, the acceptance tests created in the requirement analysis phase are executed. This testing ensures that the system is compatible with other systems. And in this, non-functional issues like:- load time, performance etc. are tested in the user environment.

V-Model (Verification and Validation)

Advantages of V-Model

- This is a simple and easy to use model.
- Planning, testing and designing tests can be done even before coding.
- This is a very disciplined model, in which phase by phase development and testing goes on.
- Defects are detected in the initial stage itself.
- Small and medium scale developments can be easily completed using it.

V-Model (Verification and Validation)

Disadvantages of V-Model

- This model is not suitable for any complex projects.
- There remains both high risk and uncertainty.
- This is not a suitable model for an ongoing project.
- This model is not at all suitable for a project which is unclear and in which there are changes in the requirement.

V-Model Example

Requirements Traceability Matrix Example:

Requirement ID	Requirement Description	Test Case ID(s)
REQ-001	Users should be able to create accounts.	TC-001, TC-002
REQ-002	Users should be able to add items to the shopping cart.	TC-003, TC-004
REQ-003	Users should be able to proceed to checkout.	TC-005, TC-006
REQ-004	The system should calculate the total cost accurately.	TC-007, TC-008

V-Model Example

Test Cases for Online Shopping Website Requirements:

Requirement: Users should be able to create accounts (REQ-001)
Test Case 1 (Positive Test):

Input: Valid user details (name, email, password).
Expected Result: User account is created successfully.

Test Case 2 (Negative Test):
Input: Existing email address.
Expected Result: System displays an error message, preventing the creation of a duplicate account.

V-Model Example

Test Cases for Online Shopping Website Requirements:

Requirement: Users should be able to add items to the shopping cart (REQ-002)
Test Case 3 (Positive Test):
Action: Add an item to the shopping cart.
Expected Result: The item is added, and the cart reflects the updated quantity and total cost.

Test Case 4 (Negative Test):
Action: Attempt to add an out-of-stock item to the cart.
Expected Result: System prevents adding out-of-stock items, and an appropriate message is displayed.

V-Model Example: System Design Phase

Background:

Initial requirements state that the online banking system should allow users to perform transactions, view account balances, and manage their accounts securely.

Architectural Design:

Activity: Define the overall structure of the online banking system.

Example Tasks:

- Identify key components: User Interface, Transaction Processing, Account Management.
- Define the relationships and interactions between components.

V-Model Example: System Design Phase

3. High-Level Design:

Activity: Break down the system into modules or subsystems.

Example Tasks:

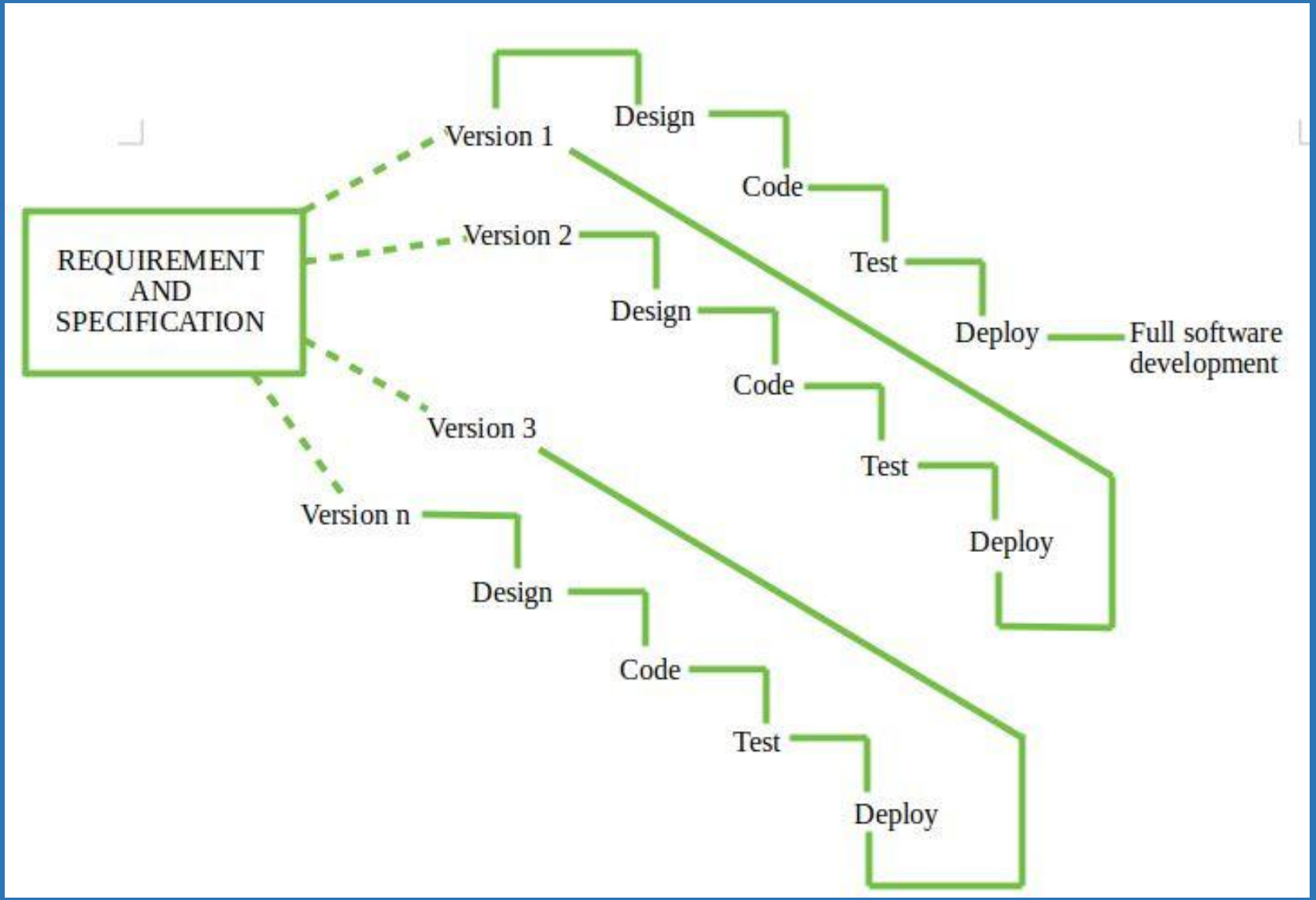
Module 1: User Authentication

- Submodule 1: Login
- Submodule 2: Logout

Module 2: Transaction Processing

- Submodule 1: Funds Transfer
- Submodule 2: Bill Payments

Incremental Model



Incremental Model

Incremental Model Case Study: Classroom Management System

Increment 1: Basic Attendance Tracking

Stakeholder Meetings:

- Meet with teachers, administrators, and other school staff to understand their needs and expectations for a classroom management system.
- Discuss the challenges they currently face in attendance tracking.

Identify High-Priority Features:

- Determine the most critical features for the initial increment.
- Example High-Priority Features:
    - Basic user authentication for teachers.
    - Attendance tracking for each class session.
    - Absence notification for parents.

Incremental Model

User Interviews:

- Interview teachers to understand their current attendance tracking practices and preferences.
- Gather insights on how they want attendance information communicated to parents.

Development and Testing

Stakeholder Review

Final Testing and Deployment

Incremental Model

Increment 2: Grade Recording and Communication

Stakeholder Meetings (Repeat):

- Conduct additional meetings with teachers and administrators to discuss the next set of features related to grade recording and communication.
- Identify high-priority features for the second increment.

User Feedback Review:

- Review feedback received from teachers during the testing of the first increment.
- Consider user suggestions and pain points in planning the second increment.

Incremental Model

Define Increment 2 Requirements:

- Document detailed requirements for the second increment, focusing on grade recording and communication features.
- Example Requirements:
    - Teachers can record grades for assignments and exams.
    - System sends automated grade reports to parents.
    - Communication features such as announcements and messaging.

Development and Testing (Repeat):

Stakeholder Review (Repeat):

Final Testing and Deployment (Repeat):

Incremental Model

Requirements Gathering: In this initial phase, the high-level requirements for the software are gathered and analyzed. These requirements serve as a foundation for the subsequent phases.

Design: Based on the gathered requirements, the software's architecture, design, and user interfaces are planned and developed. The design is often divided into smaller segments to ensure a focused and organized development process.

Implementation: Each increment involves implementing a portion of the software's functionality. Developers work on adding features, modules, or components that were planned in the design phase. This incremental approach allows for quicker delivery of usable software.

Incremental Model

Testing: As each increment is completed, testing is carried out to ensure that the new features work as expected and do not negatively impact the existing functionality. This ongoing testing helps catch and address issues early in the development process.

Integration: In this phase, the newly developed increments are integrated into the existing software. This can involve merging code, resolving conflicts, and ensuring that all components work together smoothly.

Evaluation and Feedback: After each increment, stakeholders review the functionality added and provide feedback. This feedback can be used to refine the requirements, design, and implementation of subsequent increments.

Incremental Model

Iterative Process: The software development process iterates through the above phases, gradually adding new features and improvements. With each iteration, the software becomes more robust and feature-rich.

Prototype Model

Example: Food Delivery Mobile App

Step 1: Identify Requirements

Initial Requirements:

- Users should be able to browse restaurants and menus.
- Users should be able to place orders and make payments.
- Restaurants should be able to manage their menus and receive orders.
- Delivery drivers should be able to view and accept orders for delivery.

Prototype Model

Step 2: Develop Initial Prototype

Home Screen: Displays a list of nearby restaurants and options to search or filter by cuisine.

Restaurant Page: Shows restaurant details, menu items, and prices.

Ordering System: Allows users to add items to a cart, specify delivery details, and proceed to checkout.

Restaurant Dashboard: Enables restaurant owners to manage their menus, view orders, and update order status.

Driver Interface: Provides delivery drivers with a list of available orders, order details, and navigation assistance.

Prototype Model

Step 3: Review and Feedback

- Users find the ordering process intuitive but request additional filtering options on the home screen.
- Restaurant owners appreciate the menu management features but suggest adding the ability to customize item availability.
- Drivers find the driver interface user-friendly but request real-time updates on order status.

Step 4: Refinement

Home Screen Enhancement: Add filtering options by price range and delivery time

Customization for Restaurants: Owner can set availability hours and update item availability

Real-time Updates: Implement push notifications for drivers to receive updates on order status changes.

Prototype Model

Step 5: Prototype Testing: Conduct usability testing sessions with representative users to evaluate the effectiveness of the refined prototype.

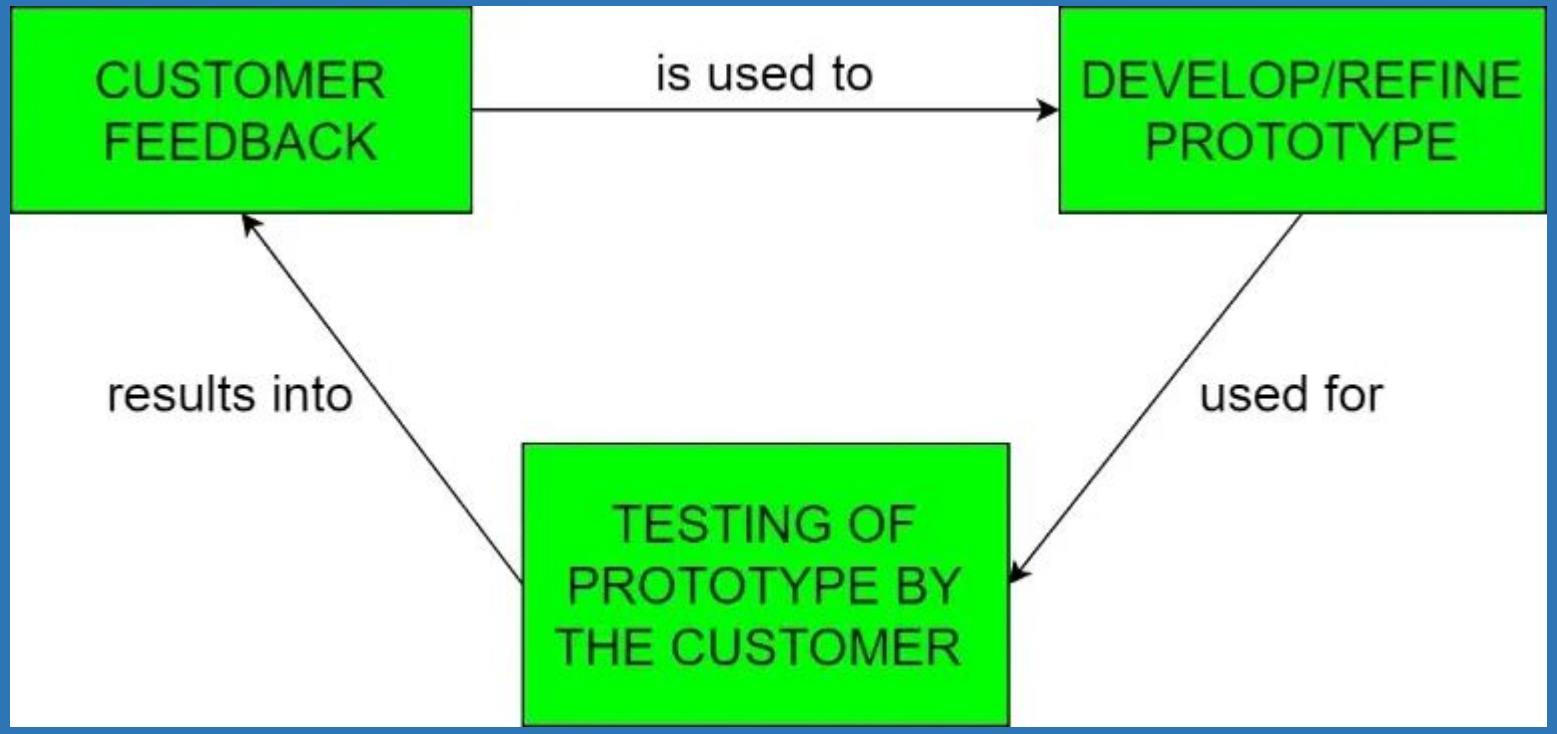
Step 6: Iterative Prototyping: Based on testing feedback, continue iterating on the prototype, making further refinements and enhancements.

Step 7: Finalization: A refined and polished prototype that incorporates all the desired features and refinements based on stakeholder feedback.

Step 8: Handoff to Development: Use the final prototype as a blueprint for developing the production-ready mobile app.

Prototype Model

The Prototyping Model is a software development methodology where a prototype (an early approximation of a final system or product) is built, tested, and then reworked until an acceptable prototype is achieved. This prototype serves as a basis for refining the final system.



Steps in the Prototyping Model

Identify Requirements:

- Gather initial requirements from stakeholders.
- Identify key functionalities and features that need to be prototyped.

Develop Initial Prototype:

- Build a basic prototype that demonstrates the core functionalities of the system.
- Focus on developing features that are essential for addressing key user needs.

Review and Feedback:

- Present the prototype to stakeholders for review and feedback.
- Gather feedback on usability, functionality, and design aspects.

Steps in the Prototyping Model

Refinement:

- Based on stakeholder feedback, refine the prototype to address identified issues and incorporate suggested enhancements.
- Make necessary adjustments to the design, functionality, and user interface.

Prototype Testing:

- Conduct usability testing and validation of the refined prototype.
- Identify any remaining issues or areas for improvement.

Iterative Prototyping:

- Repeat the prototyping process, incorporating feedback and making iterative improvements to the prototype.
- Develop multiple iterations of the prototype as needed to achieve a satisfactory solution.

Steps in the Prototyping Model

Finalization:

- Once a satisfactory prototype is achieved and stakeholders are satisfied, finalize the requirements and design specifications based on the refined prototype.

Handoff to Development:

- Use the refined prototype as a basis for developing the final system.
- Translate the prototype into production-ready code, following the established design and requirements.

Types of Prototyping Models

There are four types of Prototyping Models, which are described below.

- Rapid Throwaway Prototyping
- Evolutionary Prototyping
- Incremental Prototyping
- Extreme Prototyping

Types of Prototyping Models

1. Throwaway Prototyping (Rapid Prototyping)

Characteristics: In this approach, a basic version of the software is quickly built to demonstrate key functionalities and gather feedback from stakeholders.

Purpose: The prototype is discarded after feedback is obtained, and the final product is developed separately based on the lessons learned.

Advantages: Allows for rapid exploration of ideas and requirements validation. Can be used in situations where requirements are unclear or subject to change.

Disadvantages: May result in the loss of effort if the prototype cannot be reused in the final product. May not provide a complete solution if only focused on immediate needs.

Types of Prototyping Models

2. Evolutionary Prototyping

Characteristics: In this approach, the initial prototype is continuously refined and evolved over multiple iterations to gradually develop the final product.

Purpose: Provides a structured way to incrementally develop and refine the software based on stakeholder feedback and changing requirements.

Advantages: Allows for continuous improvement based on ongoing feedback. Provides early visibility into the evolving product.

Disadvantages: Requires careful management to avoid scope creep and maintain focus on essential features. May result in increased development time if not properly controlled..

Types of Prototyping Models

3. Incremental Prototyping

Characteristics: This approach involves developing the prototype in multiple stages, with each stage adding new features and functionalities to the existing prototype.

Purpose: Enables the development team to incrementally build and refine the software, with each iteration delivering tangible value to stakeholders.

Advantages: Provides a structured and systematic way to develop the software in manageable increments. Helps manage complexity and mitigate risks.

Disadvantages: Requires careful planning and coordination to ensure that each increment aligns with the overall project goals. May result in longer development cycles if not properly managed.

Types of Prototyping Models

4. Extreme Prototyping

This method is mainly used for web development. It consists of three sequential independent phases:

- First, an HTML prototype with all of the existing pages is displayed.
- A prototype services layer is then used to mimic data processing.
- Finally, the services are implemented and integrated into the final prototype.

Advantages

Early Validation: Stakeholders can provide feedback early in the development process, reducing the risk of costly changes later.

Improved Communication: Prototypes serve as tangible artifacts that facilitate communication between stakeholders, developers, and designers.

Flexibility: The iterative nature of prototyping allows for flexibility in accommodating changing requirements and evolving user needs.

Enhanced User Satisfaction: By involving stakeholders in the prototyping process, the final product is more likely to meet user expectations and preferences.

Disadvantages

- Prototyping is a slow and time taking process.
- Documentation is poor as the requirements change frequently.
- When the customer evaluates the prototype, there may be much too many variances in software needs.
- There is a risk of inadequate requirement analysis owing to too much dependency on the prototype.

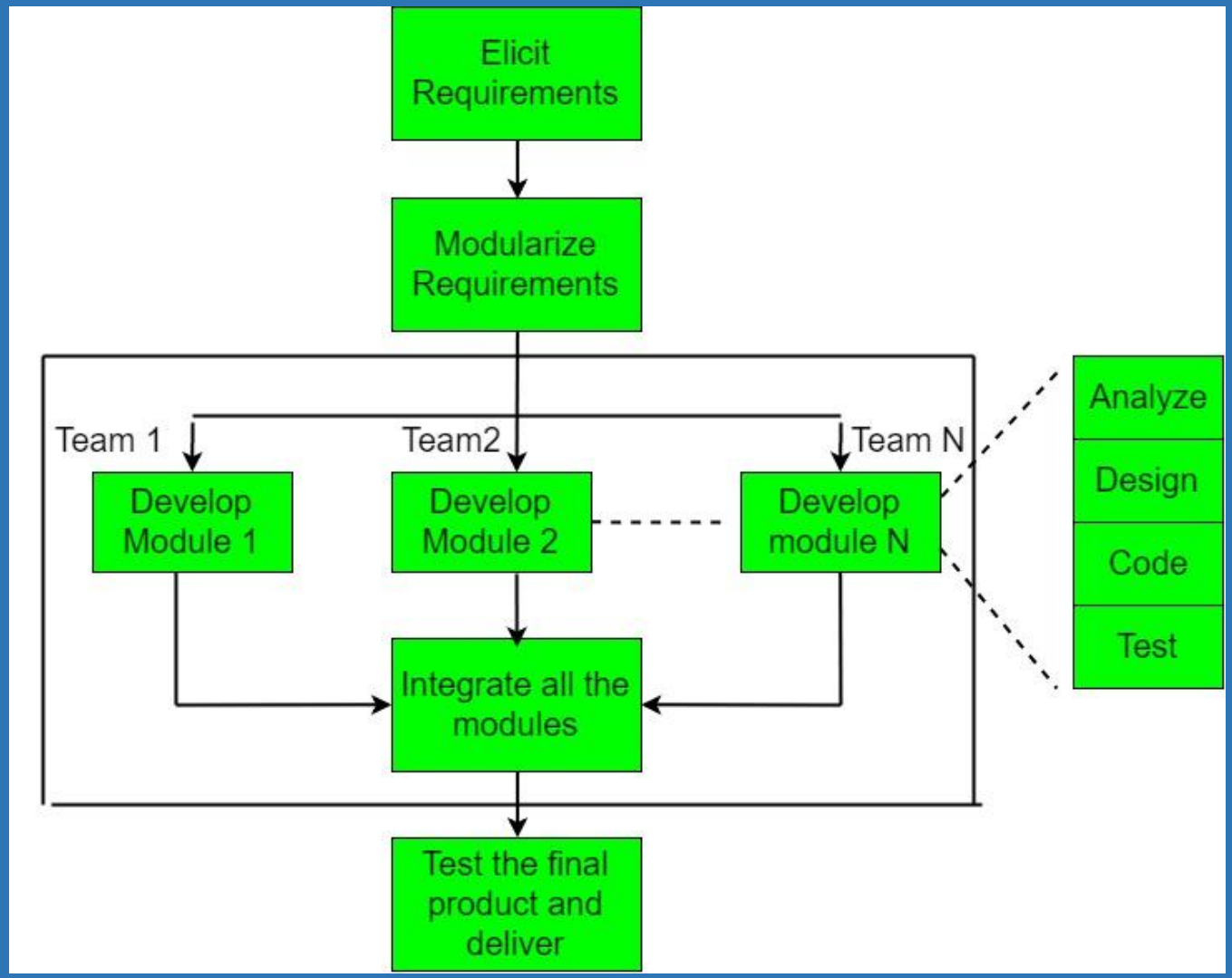
Rapid Application Development (RAD) Model

Case Study: Development of a Learning Management System (LMS) for a University

Background:

A university aims to enhance its educational offerings by implementing a modern Learning Management System (LMS) to facilitate online learning, course management, and student engagement. The university recognizes the need for a flexible and user-friendly LMS to support diverse teaching methods and accommodate the needs of both students and faculty members.

Rapid Application Development (RAD) Model



The RAD Model was first proposed by IBM in the 1980s.

The RAD model is a type of incremental process model in which there is an extremely short development cycle.

When the requirements are fully understood and the component-based construction approach is adopted then the RAD model is used.

Phases of RAD

Requirements Planning: In this phase, high-level requirements are gathered from stakeholders and prioritized based on business value. The focus is on identifying key functionalities and defining project scope.

Quick Design: Focusing on the user interface and key features. This design is iteratively refined based on feedback from stakeholders.

Construction: Development teams work in parallel to build the software components based on the design specifications. Reusable components and existing libraries are leveraged to expedite development.

Phases of RAD

Cutover: The completed software is deployed and tested in a production-like environment. User acceptance testing (UAT) is conducted to ensure that the software meets stakeholder expectations and business requirements.

Feedback and Evaluation: Feedback from end-users and stakeholders is collected throughout the development process and used to inform future iterations. Lessons learned from each iteration are applied to improve the software and development process.

Advantages of RAD

Faster Time-to-Market: RAD accelerates the development process by emphasizing rapid prototyping and iterative development, allowing for faster delivery of working software.

Increased Flexibility: RAD is highly adaptable to changing requirements and evolving user needs, thanks to its iterative and incremental approach.

Enhanced Stakeholder Collaboration: RAD promotes active involvement of stakeholders throughout the development process, leading to greater alignment between business goals and software functionality.

Improved Quality: Continuous feedback and iterative refinement result in a software product that better meets user expectations and quality standards.

Disadvantages of RAD

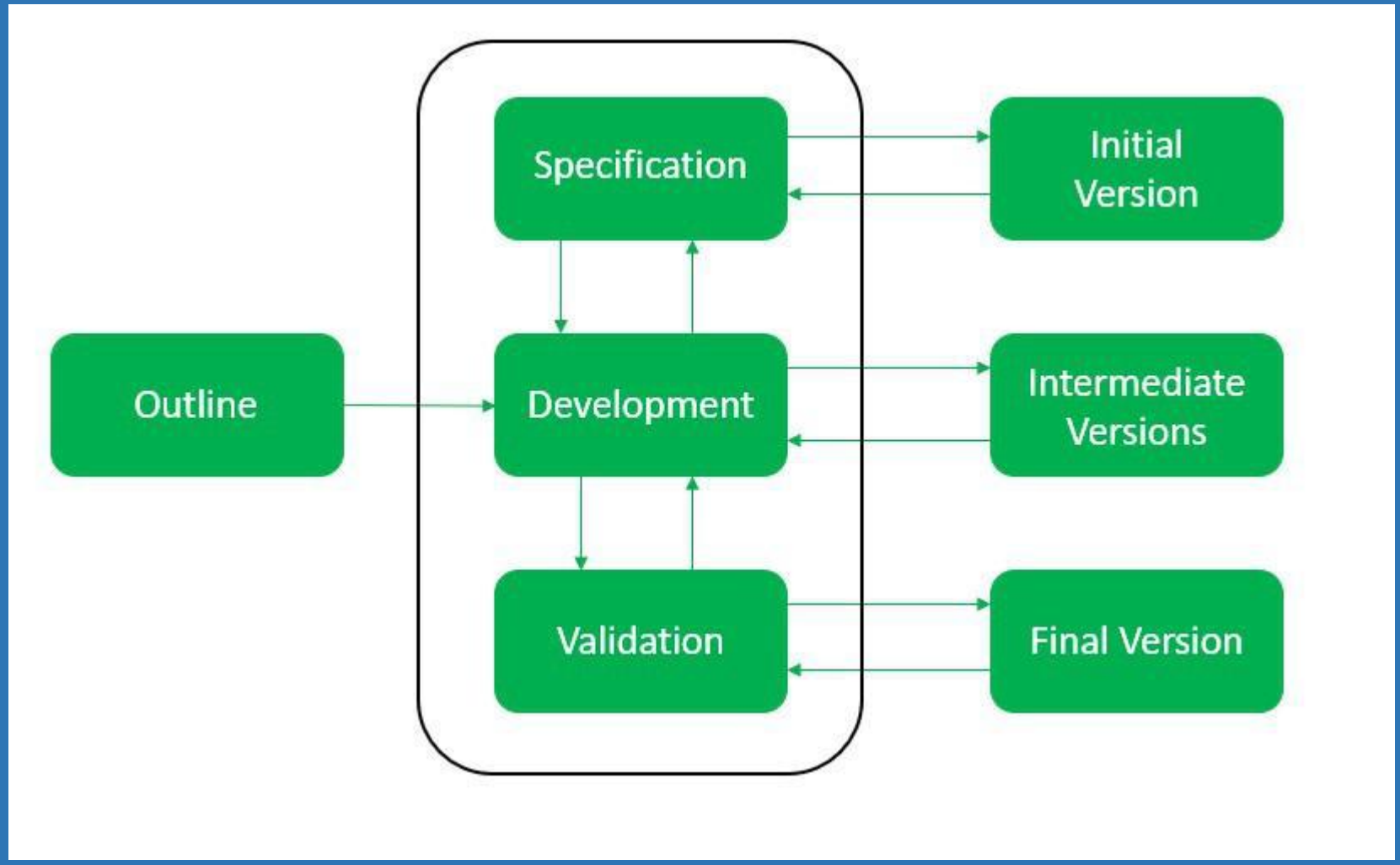
Complexity Management: Rapid iterations and parallel development can introduce complexity and dependencies that need to be carefully managed.

Resource Intensive: RAD requires significant involvement from stakeholders and development teams, which can strain resources and increase project costs.

Risk of Scope Creep: The emphasis on flexibility and rapid change can increase the risk of scope creep if requirements are not effectively managed.

Evolutionary Process Models

Evolutionary model is a combination of **Iterative and Incremental** model of software development life cycle.



Phases of the Evolutionary Model

Baseline Development: The initial version of the software, known as the baseline, is developed to establish a foundation for subsequent iterations. The baseline typically includes essential features and functionalities that address the most critical requirements.

Iterative Development: Development proceeds through a series of iterations, with each iteration adding new features or enhancements to the software. Each iteration includes activities such as requirements analysis, design, implementation, testing, and deployment.

Feedback and Evaluation: Stakeholder feedback is collected at the end of each iteration and used to evaluate the software's performance and identify areas for improvement. Lessons learned from each iteration are applied to inform future iterations and guide the evolution of the software.

Phases of the Evolutionary Model

Incremental Delivery: Working software increments are delivered at the end of each iteration, allowing stakeholders to see tangible progress and provide feedback on the evolving product. This incremental delivery approach helps maintain stakeholder engagement and confidence in the project.

Advantages of the Evolutionary Model

Flexibility: The iterative and incremental nature of the Evolutionary Model allows for flexibility in responding to changing requirements and priorities.

Continuous Improvement: Stakeholder feedback drives continuous improvement and refinement of the software, resulting in a product that better meets user needs and expectations.

Early and Ongoing Delivery: Working increments are delivered early and frequently, providing stakeholders with early visibility into the evolving product and enabling them to provide timely feedback.

Risk Reduction: By delivering working increments early, the Evolutionary Model helps mitigate project risks by allowing issues to be identified and addressed early in the development process.

Disadvantages of the Evolutionary Model

Complexity Management: Managing the evolving complexity of the software can be challenging, particularly as the number of features and iterations increases.

Resource Intensive: The iterative and incremental nature of the Evolutionary Model may require significant resources and coordination to manage multiple development cycles and parallel activities.

Dependency Management: Coordinating dependencies between different parts of the software and teams working in parallel can be complex and may require careful planning and communication.

Spiral Model

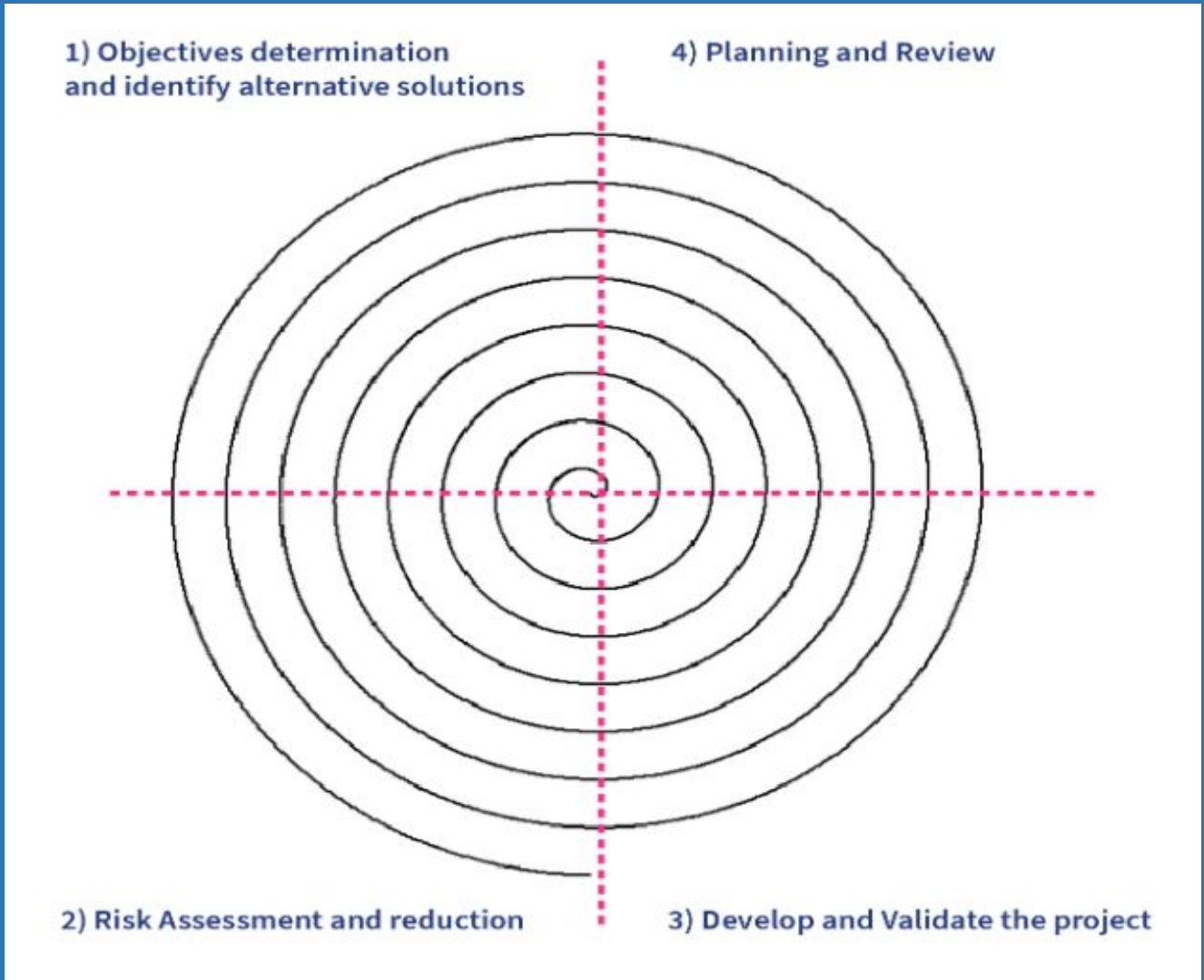
Case Study: Development of a Healthcare Information System

Background:
A healthcare organization is planning to develop a comprehensive Healthcare Information System (HIS) to streamline patient management, electronic health records (EHR), and clinical workflows.

The HIS will integrate various modules such as patient registration, appointment scheduling, medical records management, billing, and reporting.

The project is complex, with diverse stakeholders, evolving regulatory requirements, and a critical need for data security and patient privacy.

Spiral Model



- The Spiral Model is a Software Development Life Cycle (SDLC) model that provides a systematic and iterative approach to software development.
- In its diagrammatic representation, looks like a spiral with many loops. The exact number of loops of the spiral is unknown and can vary from project to project.
- Each loop of the spiral is called a Phase of the software development process.

Phases of Spiral Model

Planning: The project objectives, requirements, and constraints are established, and a development plan is created. Alternative approaches and solutions are evaluated.

Risk Analysis: Potential risks and uncertainties related to the project are identified, assessed, and prioritized. Risk mitigation strategies are developed to address high-priority risks.

Engineering: The software is designed, implemented, and tested based on the requirements identified in the planning phase. Incremental development occurs, with each iteration resulting in the delivery of a working prototype or increment of the software.

Evaluation: The completed prototype or increment is evaluated to assess its functionality, quality, and adherence to requirements. Stakeholder feedback is solicited and used to refine the software and inform subsequent iterations.

Phases of Spiral Model

Each phase of the Spiral Model is divided into four quadrants

1. Objectives determination and identify alternative solutions: Requirements are gathered from the customers and the objectives are identified, elaborated, and analyzed at the start of every phase. Then alternative solutions possible for the phase are proposed in this quadrant.

2. Identify and resolve Risks: During the second quadrant, all the possible solutions are evaluated to select the best possible solution. Then the risks associated with that solution are identified and the risks are resolved using the best possible strategy. At the end of this quadrant, the Prototype is built for the best possible solution.

Phases of Spiral Model

3. Develop the next version of the Product: During the third quadrant, the identified features are developed and verified through testing. At the end of the third quadrant, the next version of the software is available.

4. Review and plan for the next Phase: In the fourth quadrant, the Customers evaluate the so-far developed version of the software. In the end, planning for the next phase is started.

Risk Handling Spiral Model

- The spiral model supports coping with risks by providing the scope to build a prototype at every phase of software development.
- The Prototyping Model also supports risk handling, but the risks must be identified completely before the start of the development work of the project.
- In each phase of the Spiral Model, the features of the product dated and analyzed, and the risks at that point in time are identified and are resolved through prototyping.

Advantages of Spiral Model

- **Risk Management:** The Spiral Model provides a systematic approach to risk management, allowing potential risks and uncertainties to be identified, assessed, and mitigated throughout the development process.
- **Flexibility:** The iterative nature of the Spiral Model allows for flexibility in responding to changing requirements, priorities, and project conditions.
- **Early and Ongoing Evaluation:** Stakeholder feedback is solicited and incorporated throughout the development process, ensuring that the software meets user needs and expectations.
- **Adaptability:** The Spiral Model is well-suited for projects with high levels of uncertainty or complexity, as it allows for continuous refinement and adjustment based on emerging risks and evolving project needs.

Disadvantages of Spiral Model

- **Complexity:** The Spiral Model can be complex to implement and manage, particularly for large-scale projects with multiple stakeholders and dependencies.
- **Resource Intensive:** The iterative nature of the Spiral Model may require significant resources and coordination to manage multiple development cycles and parallel activities.
- **Documentation Overhead:** The Spiral Model may require extensive documentation to support risk analysis, decision-making, and stakeholder communication, which can increase overhead and project costs.